Introduction to Software Engineering

Dewayne E Perry
Office: ACE 5.124 - Hours MW 11-12:00
Phone: +1.512.471.2050
perry @ ece.utexas.edu
www.ece.utexas.edu/~perry/education/SE-Intro/
You Just Made the Wrong Choice 😊

HÅGAR
THE HORRIBLE

Hamlet, when you grow up
you'll be working for a
long, long time!

And it's important to choose a
profession that will make
you happy!

To be a success in life you have
to love what you're doing!

Of course, it would
help if what you love
to do pays a
lot of money!

Let's sit for a bit, Hamlet...

Your father has
something he
wants to tell you...

So don't base your decision
solely on making money...

© 2005-present, Dewayne E Perry
Sometimes 😊

PREPARE A PROPOSAL FOR THIS CUSTOMER.
WHY ME?

YOU WERE WALKING BY. I HAD IT IN MY HAND.

WE CAN'T WIN THIS BUSINESS. WE DON'T HAVE THE RIGHT PRODUCTS OR EXPERTISE.

JUST SAY WE DO, WE'LL FIGURE IT OUT LATER.

THEY KNOW WE DON'T. AND WE'D STILL BE THE MOST EXPENSIVE BIDDER.

BID LOW. WE'LL MAKE IT UP WITH CHANGE ORDERS AND UNEXPECTED ESSENTIAL UPGRADES.

IN OTHER WORDS, I'VE BEEN RANDOMLY ASSIGNED TO CREATE LIES FOR A PROPOSAL WE CAN'T WIN FOR A SERVICE WE CAN'T PERFORM.

YOU MAKE COMPETING SOUND BAD.
Course Information

www.ece.utexas.edu/~perry/education/SE-Intro

- Syllabus -
  - Lists papers to be read in preparation for each lecture
  - Online: at www.ece.utexas.edu/perry/education/
  - All papers are there to be downloaded

- Class is discussion!
  - Preparation: read the papers
  - Will provide study/thought questions to consider while reading
  - In class exercises

- Grades: weekly (possibly more) quizzes; 2 exams (no final exam)
  - At the beginning of class for that day’s readings
  - NO make-up quizzes - will drop lowest two scores
  - NO make-up exams except under dire circumstances
  - 90%, 80%, 70%, 60%, 50% grade structure
  - Grad students - project with incremental schedule

- Concepts and principles are the point in this course
  - Details are there to help understand the concepts and principles - will not hold you to remembering all the details
  - See the handout on how to read papers

- Sample test there to give you an idea for quizzes & exams
- Standard ECE and UT no cheating policies
Other Matters

ë Class attendance
 çer Do not take attendance - BUT will call on you to answer questions
 çer BUT weekly (or more) quizzes and two (1st half; 2nd half) exams
 çer Generally, no PPT slides - class will be devoted to discussion

ë Missing quizzes and exams
 çer You are expected to be here for tests
 çer IF you are going to miss, get to me first
 çer Has to be a significant reason
 çer There are phones with answer machines (office: 471-2050)
 çer There is email (perry @ ece.utexas.edu)
 çer And there is personal contact (I am usually around mornings)
 çer The only excuse for not getting to me ahead of time is a death in the family - yours!
 çer Interviews for jobs are not sufficient excuses. Your class comes first!!

ë You will get out of this as much as you put into it!
To Help You Do Well

- Improve comprehension
  - WSJ: report on studies for improving comprehension
  - Look at ART -
    - Go visit the Blanton Museum
    - Take an art class
  - Stimulates the part of the brain related to comprehension

- Improve retention
  - WSJ: report on study for improving retention
  - Writing longhand notes versus typing (eg on your laptop)
    - Writing longhand exercises that part of the brain associated with retention
    - Typing does not.

- If all else fails, eat dark chocolate
  - See proof on next slide
Proof of Dark Chocolate

HERE'S SOME DARK CHOCOLATE. STUDIES SHOW IT MAKES YOU THINK BETTER.

WHY ARE YOU SUDDENLY DOING THINGS THAT MAKE SCIENTIFIC SENSE, INSTEAD OF YOUR USUAL MAGICAL THINKING?

I JUST ATE THREE POUNDS OF CHOCOLATE. WOW. IT WORKS FAST.
Reading Assignments

- **Classic and seminal papers**
  - The underlying concepts and principles are critical!
  - You will be thankful when you go to interview for a software position – your interviewers will like what you can say about engineering software systems

- **I am going to be a CE/EE - why is SE relevant?**
  - Software is invading every aspect of our lives
  - For CE (and even EE) you will build software systems
  - The concepts and principles are just as relevant for CE/EE
    - All engineering is about design, measurement and evaluation etc

- **Building software systems is Fun!**
  - One of the most creative and intellectually challenging fields today
  - The papers provide examples and lessons
The Joys and Sorrows

Joys
- Sheer joy of making things
- Delight in working in a hackable medium
  - Thought stuff
  - Limits: imagination, logic and complexity
- Fashioning complex puzzle-like objects
- Creativity - grand concepts
- Always learning new things
- Making things useful for/to other people

Sorrows
- Other people often set the objectives and boundaries
- Has to work perfectly
  - Finding bugs is hard work
  - Debugging has linear convergence, or worse
  - Make progress by finding our silly and not so silly mistakes
- What we build may be obsolete before completed
We are here to learn about software engineering
We have a book and papers for basic understanding
There are libraries, internet sites, colleagues, and me to supplement your basic knowledge
Overview of Course

- Overview of Software Engineering
- Life-Cycle Phases - $\frac{1}{2}$ semester
  - Requirements
  - Architecture & design
  - Construction
  - Deployment & Maintenance
- Integral Activities - $\frac{1}{4}$ semester
  - Documentation
  - Measurement & evaluation
  - Management of objects
  - Teamwork
  - Evolution
- Process Life-Cycle & Integral activities - $\frac{1}{4}$ semester
- Project Management - week before 2nd exam
SE Life-Cycle

Product

<table>
<thead>
<tr>
<th>Phases</th>
<th>Requirements</th>
<th>Architecture &amp; Design</th>
<th>Construction</th>
<th>Deployment &amp; Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral to all phases</td>
<td>Documentation</td>
<td>Measurement &amp; Evaluation</td>
<td>Manage Objects</td>
<td>Teamwork</td>
</tr>
</tbody>
</table>

© 2005-present, Dewayne E Perry
Software Engineering (SE)

- Software Engineering is about building, maintaining and evolving software systems
  - Fundamentally, SE is a set of problem solving skills, methods, techniques and technology applied in a variety of domains to create & evolve useful software systems that solve practical problems
  - Programming is just one of these basic problem solving skills

- Brooks: “Software entities are more complex for their size than perhaps any other human construct”

- Wulf & Shaw: “Large programs, even not so large programs, are among the most complex creations of the human mind”

- Why?
  - Need more memory? Add more memory cards - replicate
  - In SE, add new distinct components, generally little replication.

- Basic Job of a Software Engineer
  - Discover, create, build and evolve
    - abstractions, behaviors and representations
  - Effectively evaluate and decide among alternative solutions
SE and Other Engineering Disciplines

- Two major components in engineering systems
  - Design
  - Manufacture

- Engineering is applied to both design and manufacture
  - Significant part of an engineering discipline is the manufacturing process
    - Have mathematics, for example, for optimization of processes
    - Engineer manufacturing and fabrication equipment

- SE: engineering is applied to both as well, BUT
  - Manufacture is
    - Trivial (by comparison - sometimes complex and time-consuming)
    - Mundane
    - Automated
  - Much larger emphasis on engineering applied to DESIGN
    - Building a software product is a DESIGN process
    - General design approaches/principles applied to diverse domains
Essential Characteristics of Software Systems

- Main Message of Brooks' *No Silver Bullet* paper:
  ... no single development, in either technology or management technique, that by itself promises even an order of magnitude improvement in productivity, reliability or simplicity!

- Brooks distinguishes between
  - Essential characteristics
  - Accidental characteristics

- Basic fact (and first important lesson):
  Building software systems is just plain hard

- Essence of software systems
  - A construct of interlocking constructs: data sets, relations between/among data, algorithms and invocations
  - Abstract
Essential Characteristics of Software Systems

Essential characteristics
- Complexity
- Conformity
- Changeability
- Invisible
- Implicit
- Evolution

Accidental Characteristics
- Inadequate modes/means of expressions
- Inadequate abstractions
- Inadequate support
- Resource limitations
Dilbert & Brooks

EVERYTHING YOU SAID IS RIGHT, BUT I HAVE A REFLEXIVE URGE TO DISAGREE WITH YOU.

IF YOU DON'T MIND, I'M GOING TO MAKE A RIDICULOUS COUNTERPOINT JUST TO GET IT OUT OF MY SYSTEM.

OKAY, BUT DON'T BE CREEPY ABOUT IT.

SOFTWARE CAN'T BE CHANGED. I AHHH... THAT'S GOOD.
Essential Characteristics of Software Systems

**Complexity**

**Basic issues**
- No two parts alike - i.e., all parts distinct
- Scale up by addition, not replication
- Very large number of states - hard to conceive, understand

**2 kinds of complexity**

**Intricacy**
- Particularly true of algorithms
- Like a Bach 4 voice fugue
  - Horizontal and vertical relationships
  - Hard to change one note without severe repercussions

**Wealth of detail**
- Nothing very deep, just masses of details
- Like a Strauss tone poem, or Mahler symphony
  - Massive number of notes on a page - provide texture
  - Missing one would hardly be noticed
- Makes very hard to comprehend the entire system (e.g., 10M lines)
Complexity: Intricacy (Bach)
Complexity: Wealth of Detail (Strauss)
Essential Characteristics of Software Systems

- Conformity
  - There are complex objects in physics
    - BUT they have uniformity
  - Not so in software systems – eg, interfaces
    - Often arbitrary complexity

- Changeability
  - Thought stuff \(\rightarrow\) infinitely malleable
  - Hence, *soft*

- Invisible
  - Not inherently embedded in space
  - No inherent geometric representation
  - Multi-dimensional relationships
Essential Characteristics of Software Systems

 Implicit

  Explicit part
    - Code - a desiccated relic of a long intellectual process
  - Very large design space
    - Narrow to code thru large number of design decisions
    - Various architectural, design and implementation decisions
    - Numerous and various trade-offs
  - Syntax represents gross and obvious dependencies
  - BUT, not the logical or semantic dependencies

 Evolution

  - Not a matter of “getting it right the first time”
    - Though sometimes that needs to be done
  - Changes in the world forces evolution
    - Context
    - Use
    - Technology
Accidental Characteristics of Software Systems

- Inadequate modes/means of expression
  - **Languages are important:**
    - Wittgenstein: “the limits of my language are the limits of my world”
    - Johnson: “language is the dress of thought”
  - **High Level Languages**
    - Frees us from accidental complexity
    - Provides useful abstractions that can be automatically checked
  - **Eg, Ada**
    - Modularity, abstraction, concurrency
    - BUT, still just an incremental improvement
  - **Eg, OO**
    - Abstract data types + hierarchical types with inheritance
    - Reduces syntactic stuff with no information content
    - BUT, type underbrush is not 9/10ths of the work we do
Accidental Characteristics of Software Systems

- Inadequate abstractions
  - AI heuristics
    - rules of thumbs
    - But much doesn’t apply
  - Graphical programming – not convincing
    - An exception: Kramer & Magee’s state simplification work
      - Helps to find faults and reduces accidental complexity
  - Automatic programming: higher level language + generator
    - Need well understood domain
    - Relatively few parameters
    - Known methods for alternatives
    - Explicit rules for selecting solution techniques
  - Program verification: verify instead of test
    - No magic – hard work
    - Programming hard, Specifications harder, proofs harder yet
      - Very hard to debug the specifications
      - Virtually all published proofs of programs have bugs
Accidental Characteristics of Software Systems

- Inadequate support
  - Programming environments
    - Libraries, structures, standard formats
  - Eg, language oriented editors
    - Never did make it
    - Useful: integrated data base for impact details

- Resource limitations
  - Time-sharing systems
    - Immediacy, availability, continuity
  - Workstations
    - Think time still dominant
  - Cloud - just servers on steroids
    - Expands availability
    - But still possible connection problems
Brooks’ Recommendations

- **Buy not build**
  - Will see later there are “flaws in the ointment”

- **Requirements, refinement, prototypes**

- **Incremental development**
  - Grow, don’t build, software systems

- **Use great designers**
  - Good design practices $\rightarrow$ good designs
    - Can be taught
  - Great designs $\rightarrow$ need great designers
    - Creative (the difference between Salieri and Mozart)
    - Achieve conceptual integrity
      - The right mix of simplicity and functionality